

Name :	Ms.Neerja Doshi
UID	2021300029
Div	SE CE DIV A – BATCH B
Experiment No. 7: Backtracking strategy – N-Queen problem	

AIM :	Backtracking strategy – N-Queen problem
THEORY :	<p>1. The N Queen is the problem of placing N chess queens on an $N \times N$ chessboard so that no two queens attack each other</p> <p>2. ALGORITHM :</p> <ol style="list-style-type: none"> Initialize an empty chessboard of size $N \times N$. Start with the leftmost column and place a queen in the first row of that column. Move to the next column and place a queen in the first row of that column. Repeat step 3 until either all N queens have been placed or it is impossible to place a queen in the current column without violating the rules of the problem. If all N queens have been placed, print the solution. If it is not possible to place a queen in the current column without violating the rules of the problem, backtrack to the previous column. Remove the queen from the previous column and move it down one row. Repeat steps 4-7 until all possible configurations have been tried. <pre> function solvenQueens(board, col, n): if col >= n: print board return true for row from 0 to n-1: if isSafe(board, row, col, n): board[row][col] = 1 if solvenQueens(board, col+1, n): return true board[row][col] = 0 return false function isSafe(board, row, col, n): for i from 0 to col-1: if board[row][i] == 1: return false </pre>

```

for i,j from row-1, col-1 to 0, 0 by -1:
    if board[i][j] == 1:
        return false
for i,j from row+1, col-1 to n-1, 0 by 1, -1:
    if board[i][j] == 1:
        return false
return true

board = empty NxN chessboard
solveNQueens(board, 0, N)

```

3. Time Complexity :

1. **Time Complexity:** $O(N!)$ where N is number of queens , and also the the number of rows and columns in given board .

CODE :

```

#define N 4
#include <stdbool.h>
#include <stdio.h>

/* A utility function to print solution */
void printSolution(int board[N][N])
{
    for (int i = 0; i < N; i++) {
        for (int j = 0; j < N; j++)
            printf(" %d ", board[i][j]);
        printf("\n");
    }
}

/* A utility function to check if a queen can
be placed on board[row][col]. Note that this
function is called when "col" queens are
already placed in columns from 0 to col -1.
So we need to check only left side for
attacking queens */
bool isSafe(int board[N][N], int row, int col)
{
    int i, j;

    /* Check this row on left side */
    for (i = 0; i < col; i++)
        if (board[row][i])
            return false;

    /* Check upper diagonal on left side */
    for (i = row, j = col; i >= 0 && j >= 0; i--, j--)

```

```

        if (board[i][j])
            return false;

        /* Check lower diagonal on left side */
        for (i = row, j = col; j >= 0 && i < N; i++, j--)
            if (board[i][j])
                return false;

        return true;
    }

    /* A recursive utility function to solve N
    Queen problem */
    bool solveNQUtil(int board[N][N], int col)
    {
        /* base case: If all queens are placed
        then return true */
        if (col >= N)
            return true;

        /* Consider this column and try placing
        this queen in all rows one by one */
        for (int i = 0; i < N; i++) {
            /* Check if the queen can be placed on
            board[i][col] */
            if (isSafe(board, i, col)) {
                /* Place this queen in board[i][col] */
                board[i][col] = 1;

                /* recur to place rest of the queens */
                if (solveNQUtil(board, col + 1))
                    return true;

                /* If placing queen in board[i][col]
                doesn't lead to a solution, then
                remove queen from board[i][col] */
                board[i][col] = 0; // BACKTRACK
            }
        }

        /* If the queen cannot be placed in any row in
        this column col then return false */
        return false;
    }

    /* This function solves the N Queen problem using
    Backtracking. It mainly uses solveNQUtil() to
    solve the problem. It returns false if queens
    cannot be placed, otherwise, return true and
    prints placement of queens in the form of 1s.
    Please note that there may be more than one
    solutions, this function prints one of the

```

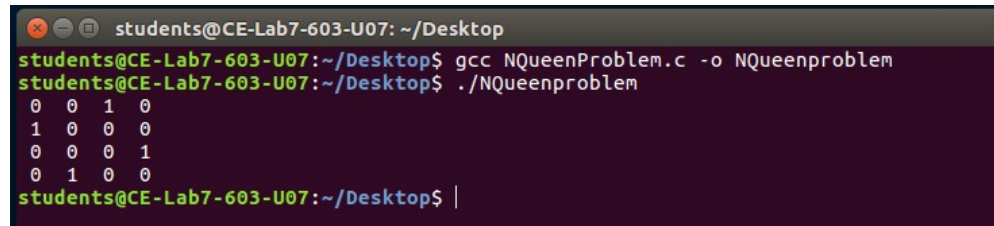
```
feasible solutions.*/
bool solveNQ()
{
    int board[N][N] = { { 0, 0, 0, 0 },
                          { 0, 0, 0, 0 },
                          { 0, 0, 0, 0 },
                          { 0, 0, 0, 0 } };

    if (solveNQUtil(board, 0) == false) {
        printf("Solution does not exist");
        return false;
    }

    printSolution(board);
    return true;
}

// driver program to test above function
int main()
{
    solveNQ();
    return 0;
}
```

OUTPUT :



```
students@CE-Lab7-603-U07: ~/Desktop
students@CE-Lab7-603-U07:~/Desktop$ gcc NQueenProblem.c -o NQueenproblem
students@CE-Lab7-603-U07:~/Desktop$ ./NQueenproblem
0 0 1 0
1 0 0 0
0 0 0 1
0 1 0 0
students@CE-Lab7-603-U07:~/Desktop$ |
```

CONCLUSION : By performing the above experiment , i have successfully understood to perform Backtracking by solving N – Queen Problem.